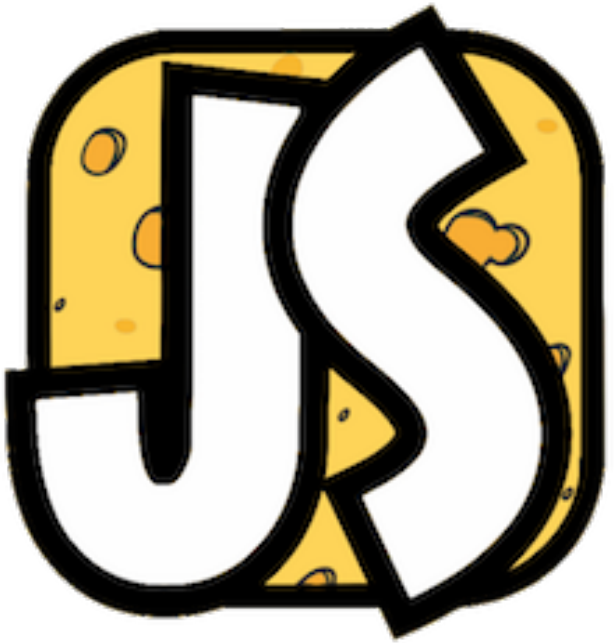
Towards N-API on JerryScript

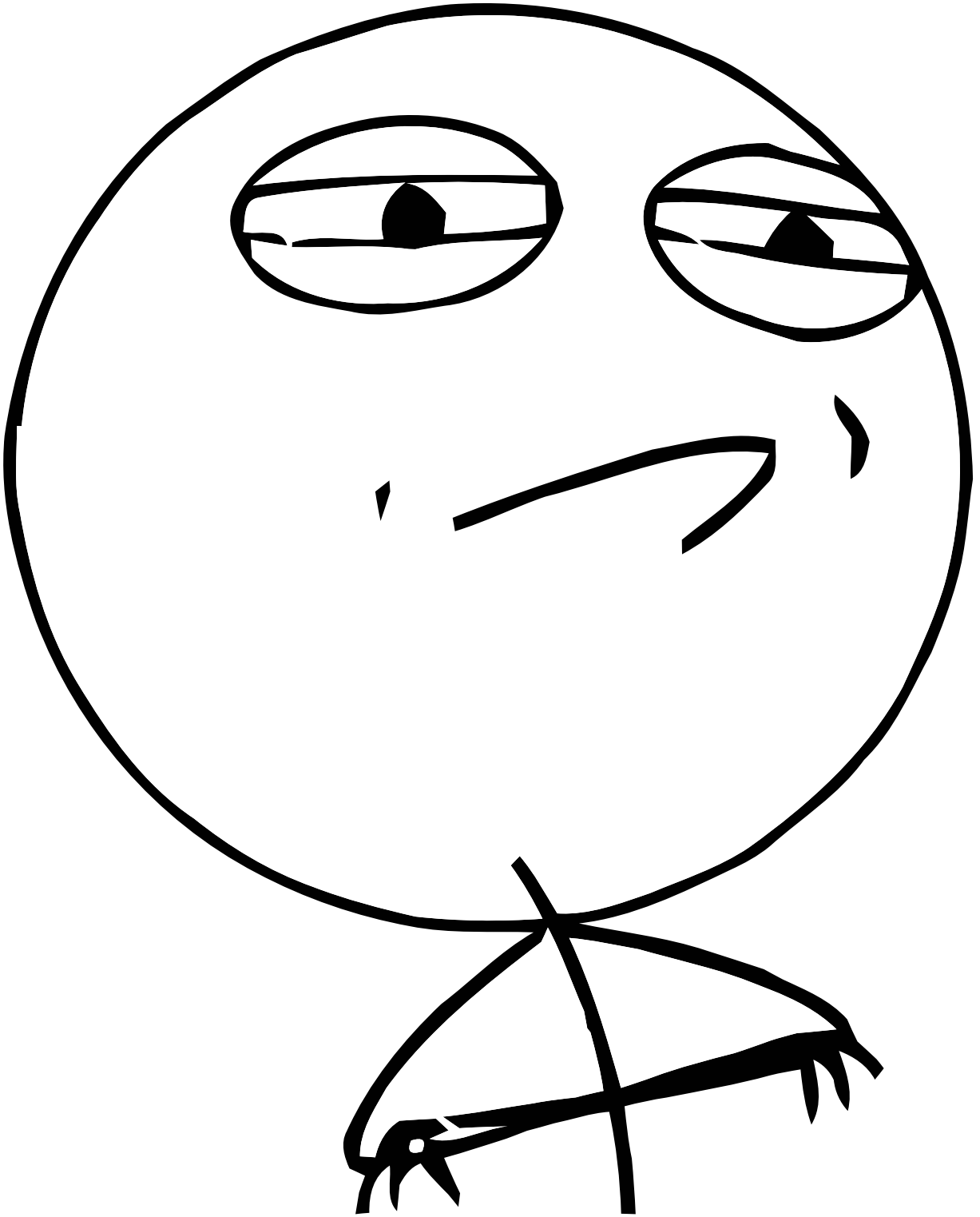


Gabriel Schulhof <gabriel.schulhof@intel.com>
@gabrielschulhof

intel

Intro

- Emphasis on low memory consumption
- Able to run with less than 64 KiB of RAM
- Takes up less than 200 KiB of storage
- Challenges
  - Every byte counts
  - One does not simply `malloc()`
- Goal: Implement N-API, so N-API modules can use a single code base on both Node.js and JerryScript



Step 0

- Where to put `napi_env`?
  - In Node.js it's currently per-module, but we're working on per-context
  - JerryScript already has a context stored thread-locally and not passed around so as to reduce stack usage
- Where to put loaded module instances for `require()` caching?
- Solution: A mechanism for attaching data to the context

```
// API

typedef struct {
  void (*init_cb) (void *);
  void (*deinit_cb) (void *);
  size_t bytes_needed;
} jerry_context_data_manager_t;

void *
jerry_get_context_data (const jerry_context_data_manager_t *manager_p);

// Usage

static const jerry_context_manager_t
my_context_data_manager = {
  .init_cb = my_context_data_init,
  .deinit_cb = my_context_data_deinit,
  .bytes_needed = sizeof(my_context_data_t)
};
...
my_context_data_t *my_context_data =
    jerry_get_context_data(&my_context_data_manager);
...
```

Step 1

- Ummm … modules?
- Two levels:
  - Resolvers
    - Module definitions
- Resolvers know how to load modules
  - Text modules
  - Binary modules
- Default module definition provided – via linker section

```
typedef jerry_value_t (*jerryx_native_module_on_resolve_t) (void);

typedef struct
{
  jerry_char_t *name;
  jerryx_native_module_on_resolve_t on_resolve;
} jerryx_native_module_t;

#define JERRYX_NATIVE_MODULE(module_name, on_resolve_cb) \
  static const jerryx_native_module_t _module                \
    JERRYX_SECTION_ATTRIBUTE(jerryx_modules) =                \
  {                                                           \
    .name = ((jerry_char_t *) #module_name),                  \
    .on_resolve = (on_resolve_cb)                             \
  };

bool
jerryx_module_native_resolver (const jerry_char_t *name,
                               jerry_value_t *result);
typedef bool (*jerryx_module_resolver_t) (const jerry_char_t *name,
                                          jerry_value_t *result);
jerry_value_t
jerryx_module_resolve (const jerry_char_t *name,
                       jerryx_module_resolver_t *resolvers,
                       size_t count);
```

Remaining

- Engine
  - Symbols → JerryScript is so far only up to ES 5.1
  - `ArrayBuffer` and `TypedArray`
- N-API implementation
  - Externals
  - Scope → JerryScript values are released explicitly
  - Multiple string encodings
  - Async work via job queue
  - `Buffer`

Thank you!

Gabriel Schulhof <gabriel.schulhof@intel.com>
@gabrielschulhof